MORRISON & FOERSTER LLP
MICHAEL A. JACOBS (Bar No. 111664)
mjacobs@mofo.com
KENNETH A. KUWAYTI (Bar No. 145384)
kkuwayti@mofo.com
MARC DAVID PETERS (Bar No. 211725)
mdpeters@mofo.com
DANIEL P. MUINO (Bar No. 209624)
dmuino@mofo.com
755 Page Mill Road, Palo Alto, CA 94304-1018
Telephone: (650) 813-5600 / Facsimile: (650) 494-0792

BOIES, SCHILLER & FLEXNER LLP
DAVID BOIES (Admitted *Pro Hac Vice*)
dboies@bsfllp.com
333 Main Street, Armonk, NY 10504
Telephone: (914) 749-8200 / Facsimile: (914) 749-8300
STEVEN C. HOLTZMAN (Bar No. 144177)
sholtzman@bsfllp.com
1999 Harrison St., Suite 900, Oakland, CA 94612
Telephone: (510) 874-1000 / Facsimile: (510) 874-1460

ORACLE CORPORATION
DORIAN DALEY (Bar No. 129049)
dorian.daley@oracle.com
DEBORAH K. MILLER (Bar No. 95527)
deborah.miller@oracle.com
MATTHEW M. SARBORARIA (Bar No. 211600)
matthew.sarboraria@oracle.com
500 Oracle Parkway, Redwood City, CA 94065
Telephone: (650) 506-5200 / Facsimile: (650) 506-7114

Attorneys for Plaintiff
ORACLE AMERICA, INC.

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, v. GOOGLE INC. Defendant.	Case No. CV 10-03561 WHA **ORACLE'S MAY 10, 2012 BRIEF RESPONDING TO COURT'S QUESTIONS ON COPYRIGHTABILITY** Dept.: Courtroom 8, 19th Floor Judge: Honorable William H. Alsup

1. If the Copyright Act is meant to protect expression but not vocabulary, should the vocabulary and grammar of a computer language be copyrightable, as distinct from programs written in the language? In this regard, please comment on the May 2, 2012, decision of the High Court of the European Union.

Protectability of a computer programming language is not the issue in this case. However, if sufficiently original and creative, the vocabulary and grammar of a computer language may be protected by copyright. In *Reiss v. National Quotation Bureau, Inc.*, Judge Learned Hand upheld copyright protection for a code book containing coined words that could be assigned an agreed meaning for use in cable correspondence. 276 F. 717, 718-19 (S.D.N.Y. 1921). Similarly, in *Hatfield v. Peterson*, the Second Circuit upheld copyright protection for a telegraphic code that was a compilation of non-original words and phrases, holding "the copyright is valid because of the originality of the combination." 91 F.2d 998, 1000 (2d Cir. 1937). An individual word alone is not protectable, but if sufficiently original, the selection and combination of a collection of vocabulary words in a programming language should be protectable under these same principles.

Although the threshold of originality may be more difficult to meet, the same applies to the grammar evaluated in combination with that vocabulary. The grammar potentially adds to the originality of the combination and arrangement of the words and, if sufficiently creative, merits protection. *See Toro Co. v. R & R Prods*, 787 F.2d 1208, 1213 (8th Cir. 1986) ("A system that uses symbols in some sort of meaningful pattern, something by which one could distinguish effort or content, would be an original work."). (*See also* ECF No. 900 at 1-5.)

The recent decision of the Court of Justice of the European Union ("CJEU") states that programming languages might be protected as copyrighted works if they are their author's own intellectual creation. Case C-406/10 *SAS Institute, Inc. v. World Programming Ltd.*, Judgment (May 2, 2012) ¶ 45. The *SAS* case was a reference, which asked for guidance from the CJEU on a series of questions. The first set of questions concerned copyright protection for programming languages, the functionality of a computer program and the format of data files. *SAS* ¶ 29.

In *SAS*, the CJEU addressed the potential copyrightability of these components under two EU "directives:" Directive 91/250 (the "Software Directive") and Directive 2001/29 (the "Copyright Directive"). The Software Directive applies to computer programs (*see id.* ¶ 8(7),

9(2)); the Copyright Directive has more general application. *See id.* ¶ 15 (2). The court concluded that a programming language is not protected under the Software Directive: "neither the functionality of a computer program nor the programming language and the format of data files used in a computer program to exploit certain of its functions constitute a form of expression of that program for the purposes of Article 1(2) of Directive 91/250." *Id.* ¶ 39.[1]

A programming language *could* be copyrightable under the Copyright Directive, however:

> The Court also points out that the finding made in paragraph 39 of the present judgment cannot affect the possibility that the SAS language and the format of SAS Institute's data files might be protected as works by copyright under Directive 2001/29 if they are their author's own intellectual creation.

Id. ¶ 45 (citation omitted). The CJEU decision thus concludes computer programming languages may be copyrightable if they are sufficiently original. As the Court may recall, Advocate General Bot had recommended, in a non-binding opinion, that the CJEU find that programming languages were not copyrightable. *SAS Institute, Inc. v. World Programming Ltd.*, Case C-406/10 (Nov. 29, 2011) ¶ 75. (*See* ECF Nos 852, 859). The CJEU declined to adopt that part of his opinion.

The portion of the *SAS* case that is more analogous to the present circumstances, however, concerns the set of questions relating to the defendant's copying of the SAS user manual. The CJEU held that copying from the SAS user manual into defendant's own user manual and computer program could also constitute infringement under the Copyright Directive:

> Consequently, in the light of the foregoing considerations, the answer to Questions 8 and 9 is that Article 2(a) of Directive 2001/29 must be interpreted as meaning that the reproduction, in a computer program or a user manual for that program, of certain elements described in the user manual for another computer program protected by copyright is capable of constituting an infringement of the copyright in the latter manual if—this being a matter for the national court to ascertain—that reproduction constitutes the expression of the intellectual creation of the author of the user manual for the computer program protected by copyright.

Id. ¶ 70.

Notably, the court found that, while elements described in the SAS manual—including

[1] *SAS* was only released recently and commentators are still analyzing it. It is unclear whether the court found programming languages are not covered by the Software Directive because they are not "computer programs" as the Directive requires (*see id.* ¶¶ 8(7), 9(2)) or because, while they are computer programs, they lack sufficient "expression" within the meaning of that Directive. The decision does state that the code relating to the programming language would be protectable under the Software Directive as a computer program. *See id.* ¶ 43.

keywords, syntax and commands—could not be copyrighted individually, their "choice, sequence and combination" may warrant copyright protection as an intellectual creation of the author:

> It is only through the choice, sequence and combination of those words, figures or mathematical concepts that the author may express his creativity in an original manner and achieve a result, namely the user manual for the computer program, which is an intellectual creation (see, to that effect, *Infopaq International*, paragraph 45.)

Id. ¶¶ 66-67.[2] Although *SAS* was decided under different law and involved very different facts (including no finding that the defendant had copied the structural design of the code (*id.* ¶ 25) and no issue about APIs), the CJEU decision supports a finding that Google's copying of the SSO set forth in Oracle's API documentation constitutes infringement.

2. If each API method is a program — pre-packaged but nonetheless a program — did Google copy anything more than the name and the declaration (substituting its own implementation)?

Yes. Google did not just copy the name and the declaration for each method. Google also copied the SSO of the 37 API packages. The method names and declaration it copied are an essential part of that SSO. (In this trial the term "method declaration" has often been used when "method header" is technically correct. Oracle will continue to use "declaration" in this brief.)

The method declarations Google copied include considerable original expression. The elements it copied include the mode (e.g. public or protected), the name, the parameter list, and the throws clause (exception list) of thousands of methods. (*See Java Language Specification* 3d Ed. ("JLS") § 8.4, TX 984 at 209-237 (describing components of method header. *See also* RT 1238:13-1239:12 (describing creativity in method declarations), 1249:2-12 (Mitchell).) Google copied all this into Android, including elements not required for "compatibility:"

Q. And roughly, what percentage of copying did you observe in your work?

---

[2] In the *Infopaq* case, the CJEU held the reproduction of extracts consisting of *eleven words of text* by a news summary service (5 words on either side of a key search word) fell within the scope of protection of the Copyright Directive. Case C-5/08 *Infopaq International A/S v. Danske Dagblades Forening*, [2009] ECR I-6569 (Attachment A). "[T]he possibility may not be ruled out that certain isolated sentences, or even certain parts of sentences in the text in question, may be suitable for conveying to the reader an element which is, in itself, the expression of the intellectual creation of the author of that article." *Id.* ¶ 47. That was particularly true since the defendant's data capture process "reproduces an extract of 11 words each time a search word appears in the relevant work" and allows clients to request more than one search word. *Id.* ¶ 49.

> A. Quantitatively, approximately 90 percent. There's – occasionally there's something that's not copied, and occasionally there may be some small differences in the way things are named.
>
> For example, the methods have parameters. Sometimes the parameter names are different. *Surprisingly, they're the same in probably two-thirds of the cases, even though that's not necessary for the library to be used in the same way by programmers.*

(RT 1248:11-1249:12 (Mitchell).) Google did not rebut this evidence at trial.

But this case is not based on the copying of individual methods. It is undisputed Google copied thousands of methods and other API elements and all of the relationships among them, replicating the structure, sequence and organization of Oracle's documentation and source code in the 37 packages. Google's expert concedes that this is the case. (RT at 2214:6-9 (Astrachan).)

The methods are a key part of that structure. Google chose to repeatedly emphasize a trivial example at trial, java.lang.Math.max. This method is unusually simple because (1) like other basic mathematical functions in the java.lang.Math class, it seems obvious to include it; (2) it does not depend at all upon any other methods but rather stands alone; and (3) it does not access data stored in the fields of any member of any class. It creates the misleading impression that the only creativity in method design is selecting the class and package in which to put it in.

In the more typical example, however, the methods are very much the product of creative design choices themselves. When an API designer sets out to design a new package, he or she faces many different possibilities for addressing the particular technical challenges involved. (*See* RT at 621:7-623:21 (Reinhold) (describing design process); 627:21-629:6 (discussing design choices).) There is no simple list of standalone functions like java.lang.Math.max to plug in. As Dr. Reinhold testified java.lang.Math.max does not accurately depict the work that goes into designing APIs. (*See* RT at 2228:2-16 ("max is a pretty trivial problem. In nio we had to solve a very hard problem, a portable high-performance IO API. And to do that we needed to design hundreds of methods, most of them much more complicated than max, organized into doesn't of Classes across a handful of different Packages.").)

Methods are designed to work with other methods and classes in a way that is intuitive, effective, and best suits the goal of the API design. The design solution determines the methods and the methods reflect the structure of the design.

To give a very simple example, suppose you have a Screen class such that a member (or "instance") of the class represents the display screen. If you want to be able to draw a rectangle, one possibility would be to create a method called "drawRect that takes an x, y coordinate, a height and width and draws the rectangle. The API might look like this:

```
public class Screen {
        public void drawRect(in x, int y, int width, int, height) {  ...  }
   }
```

To draw a rectangle with this API you would first create an instance of the class, using the "new" keyword and then invoke the drawRect method, like so:

```
Screen s = new Screen();       /* Get a Screen object */
s.drawRect(10, 10, 10, 10);    /* Draw a 10x10 rectangle */
```

This code would produce a rectangle that is 10x10 pixels starting at the point 10 pixels down and 10 in from the top left of your screen.

A fundamentally different choice would be to create an API that allows the user to "draw" the rectangle using a "pen." This design would include a different set of methods associated with it. The API could look like this:

```
public class Screen{
      public void dropPen(int x, int y) { ...  }
      public void raisePen() {... }
      public void turn (int degrees) {... }
      public void draw(int length) { ... }
  }
```

One way to draw the same rectangle, would be to do the following:

```
Screen s = new Screen();
s.dropPen(10, 10);
s.turn(90);  s.draw(10);
s.turn(90);  s.draw(10);
s.turn(90); s.draw(10);
s.turn(90); s.draw(10);
s.raisePen();
```

In this highly simplified example, neither API is inherently better than the other. The first is simpler, the second allows drawing other types of shapes. The methods used reflect the design

choice that was made. They do not arise inevitably from the decision to include a method for drawing a rectangle in the API.

The Court can view an actual example from Java in the method detail in TX 610.2 for java.util.regex.Matcher.appendReplacement. (TX 610.2 at /docs/api/java/util/regex/Matcher.html.) This method "[i]mplements a non-terminal append-and-replace step" within a class for matching patterns of text. Examining the code alone, the method header seems straightforward enough, taking two parameters and returning a value:

public Matcher appendReplacement(StringBuffer sb, String replacement)

Id. But this seemingly simple header is followed by several paragraphs of specification text to explain how to use it and how it works with other methods in the Matcher class, with the Pattern class elsewhere in the java.util.regex package, and with the StringBuffer class in the java.lang package. The specification of the method includes 8 lines of example code to tell developers how to use the method. The simplicity of this method's design was hardly preordained—it was the result of concerted, creative effort.

Another useful example is to consider the different approach that Oracle and Google took in designing APIs for graphical user interfaces. If a Java developer wanted to create a dialog box with a simple message and a button marked "OK," she might use the method

javax.swing.JOptionPane.showMessageDialog:
JOptionPane.showMessageDialog(frame, "This text will be displayed", "Message");

See TX 610.2 at /docs/api/javax/swing/JOptionPane.html. Displaying a similar dialog box in Android would require the use of the android.app.AlertDialog.Builder API, which requires, among other things, separate steps for creating the dialog box, setting the message to be displayed (AlertDialog.Builder.setMessage()), setting the text to display on the button (AlertDialog.Builder.setNegativeButton()), and showing the dialog (AlertDialog.create()). See TX 767 at /android/app/AlertDialog.Builder.html.

As these examples show, method names, headers, and declarations are an integral part of the structure and themselves reflect API developer's design choices. Their selection, structure, sequence and organization is original and is protectable by copyright.

3. If the format of the name is dictated by the Java programming rules, then is the form of "java.package.class.method" required by the syntax of the language itself?

Yes, although the format of the names does not dictate an API designer's choice of any particular structure, the format of fully-qualified method names must conform with the syntax of the language. But the fully qualified name can be richer than the example given in the question. A package name may contain an arbitrary number of elements, separated by periods. For example "java.lang" is a package name, and so is "java.util.concurrent.locks". (*See* TX 610.2 at /jdk-1_5_0-doc/docs/api/overview-summary.html (list of packages).) Package names need not start with "java," so "org.w3c.dom" is a package name as well. *Id.* Some of the packages that Android did not copy from Java begin with "android" instead of "java." (*See* TX 767 at packages.html.; TX 984 at 154).

There are also different structural possibilities permitted within the syntax. For example, Java allows nested "Member" classes, such as "java.nio.channels.Pipe.SinkChannel," in which one class, SinkChannel, is enclosed within another class, Pipe. (*See* TX 984 at 210; TX 610.2 at /docs/api/java/nio/channels/Pipe.SinkChannel.html; TX 623 at \licenseebundles\source-bundles\jdk-1_5_0-fcs-src-b64-windows-15_sep_2004.zip.zip\j2se\src\share\classes\java\nio\channels\Pipe.java, lines 69-98.) In the example above, the platform designers chose to declare the SinkChannel class within the source code file for the Pipe class, and that design choice is reflected in the fully qualified name.

Java allows Member interfaces, where an interface is defined within another class or interface. This is also reflected in the fully-qualified name. (*See, e.g.*, TX 984 at 237; TX 610.2 at /jdk-1_5_0-doc/docs/api/java/lang/Thread.UncaughtExceptionHandler.html (documentation for interface java.lang.Thread.UncaughtExceptionHandler, which is enclosed in the file Thread).)

4. Could Google have come up with different names and SSO yet still have provided the same functionality as in Android? Android users would have had to learn a new vocabulary and a new outline but the methods would all still have been resident in such a modified Android. True? Is this what the UK company Spring did?

Yes, but with the caveat that if Google had done this one would not expect that "the methods would all still have been resident in such a modified Android." That may be true for

very simple methods (e.g., Math.max), but more sophisticated methods would be designed for a completely different set of classes, interfaces, and packages. The high-level functionality could be the same as the copied Java APIs, but the actual set of methods would be quite different.

Both parties' experts testified that Google could have and did come up with its own APIs for Android with different names and SSO but with similar functionality to the Java APIs. (*See* RT 2288:6-12 (Mitchell); RT 2212:25-2213:19, 2220:1-7 (Astrachan).) *See also* TX 767 at packages.html (listing package names beginning with "android" compared to those beginning with "java" and "javax").)

In addition, Dr. Reinhold described an open source API and class library for much the same functionality as one of the Java API packages at issue, java.util.logging, but with a very different SSO. (RT at 630:11-631:18(Reinhold).) Mr. Screven testified there are also alternate Java APIs for mathematics and encryption. (RT at 518:4-519:15 (Screven).) Both of these programming areas are covered by the Java API packages at issue here. (*See* TX 1072 (listing logging, encryption and math packages among 37 accused packages).

The Court is correct that this is what SpringSource did:

> There's a company in the UK that built its own Java environment. And they used the Java programming language, but they created their own set of APIs, prewritten programs. And that other environment is called Spring.
>
> So Spring uses the Java programming language, but it doesn't use the Sun-created APIs. They have their own set of APIs and their own set of prewritten programs.

(RT 290:25-291:6 (Ellison).) Of course, creating new APIs was much more time consuming and expensive than what Google did here in its rush to bring Android to market:

> Q. Does it take a period of time and expense and resources if you're going to go that route?
>
> A. Yeah. Spring had to design their own APIs, and then they had to teach the developer community about these new APIs. And they had to persuade them that their collection of APIs, their library of programs, was in some ways better than the library of programs that Oracle and Sun had produced.

(*Id.* at 304:16-22.)

> **5. Is the input-output (*i.e.*, argument and return) scheme of a method copyrightable? For example, can someone copyright the function of inputting an angle and outputting the cosine of that angle? If someone has a copyright on a**

particular program to find cosines, does that copyright cover all other implementations that perform the identical function (input = angle, output = cosine)?

No, the input-output scheme for an individual method is not copyrightable. The SSO of the API packages, which includes thousands of these individual methods, is expressive and protectable in its own right. Courts have also held that a "a combination of unprotectable elements is eligible for copyright protection only if those elements are numerous enough and their selection and arrangement original enough that their combination constitutes an original work of authorship." *Satava v. Lowry*, 323 F.3d 805, 811 (9th Cir. 2003). Specifically, courts have held that combinations of original data elements in software may be copyrightable. *See Merch. Transaction Sys. v. Nelcela, Inc.*, 2009 U.S. Dist. LEXIS 25663, at *58 (D. Ariz. Mar. 17, 2009) (finding the "selection, coordination, and arrangement of the information contained in the Lexcel software's database schema (non-literal elements) and source code (those involving copying of table and column names)" constituted copyrightable subject matter.)

Oracle selected thousands of methods and gave them a complex, highly original SSO by placing them in hundreds of classes within dozens of packages. (*See* RT 602:4-603:6, 628:22-629:6 (Reinhold); RT 1248:11-1249-1 (Mitchell).) Google copied the SSO in virtually identical fashion, even where Oracle provided different variants for the method. The class java.nio.IntBuffer, for instance, has 4 different variants of the "get" method:

public abstract int get ()

public abstract int get (int index)

public IntBuffer get (int[] dst)

public IntBuffer get (int[] dst, int dstOffset, int intCount)[3]

Google copied each of these different versions of the methods as well. (Compare TX 610.2 at /jdk-1_5_0-doc/docs/api/java/nio/IntBuffer.html with TX 767 at java/nio/IntBuffer.html).

[3] The parameter names in Java for this version of the "get" method are (int[] dst, int offset, int length). As noted above, Google chose to copy Oracle's parameter names about two-thirds of the time. (RT 1248:11-1249:12 (Mitchell).)

6. Is it agreed that the following is true, at least as of 1996?

The following were the core Java Application Programming Interface: java.lang, java.util and java.io.

The record does not reflect a precise definition of the term "core." If "core" means "containing some classes mentioned in the initial version of the *Java Language Specification*," the answer is yes. But "core" should not be interpreted to mean that all of the contents of these packages are necessary to the programming language. That is not the case. According to Dr. Reinhold's analysis, only 61 classes are mentioned in the normative text of the *Java Language Specification* and, except for java.lang.Object, which is only partially specified, none is specified in detail in the Language Specification. (TX 1062; RT 676:14-678:13, 684:16-685:2 (Reinhold); RT 2196:1-4 (Astrachan), TX 984 at 6.) Dr. Bloch agreed only 60 classes are mentioned in the *Java Language Specification*, but never stated what they were. (RT at 777:21-24 (Bloch). Of the classes mentioned, almost all are in java.lang. TX 1062 (listing 55 classes from java.lang, 4 annotations from java.lang.annotation, 1 interface from java.io, and 1 interface from java.util).

These packages have all greatly expanded since they were first released in 1996. Dr. Mitchell testified at trial as to how the structure of java.util has changed over time as that package has grown dramatically in size. (RT at 1243:13-1244:16 ("It was just a smaller library and there were fewer concepts in it.").) A comparison of TX 2564 and TX 610.2 shows that java.util had only 14 classes, and 142 methods in 1996 compared to 86 classes and 1059 methods in version 5.0. Similarly, java.io grew from only 31 classes and 304 methods in 1996 to 78 classes and 753 interfaces in version 5.0. And java.lang grew from 64 classes and 462 methods in 1996 to 95 classes and 1089 methods today.

The java.util package includes classes and interfaces for events, properties, and collections (i.e., data structures such as lists, sets, and maps). java.lang includes classes for spawning and managing programs running in other operating-system processes. The java.io package includes not just basic input/output facilities but also classes for converting arbitrary Java data objects to and from streams of bytes which can be stored in a file or transmitted over a network. No experienced developer would expect such facilities to be part of a programming language *per se*.

At the hearing on May 9, the Court suggested the Java programming language may not include concepts that are more commonly found in other languages. Nothing in the record supports this. The only evidence at trial comparing the contents of the Java language to the contents of other programming languages was the testimony of Dr. Reinhold who, at the Court's request, compared the enormously popular C programming language "to the Java programming language, ignoring all of the APIs." (RT at 640:5-24.) Dr. Reinhold stated that the Java programming language is much richer, like comparing a rocket to "a DC-3." (RT at 640:18-24.)

7. Does the Java programming language refer to or require any method, class or package outside the above three?

The package java.lang.annotation was added to the platform in J2SE 5.0 There are four annotations from java.lang.annotation that are referenced in the third edition of the *Java Language Specification*, TX 984, the edition of the JLS which corresponds to Version 5. There is only one interface from java.io and one class from java.util that are referenced in TX 984. All of the remaining classes come from the java.lang package. (*See* TX 1062.)

8. To what extent do subparts of the above three Java packages refer to any other Java packages or subpart of other packages (meaning outside the three)? To the extent this occurs, should those outside subparts be deemed to be "core" to the programming language?

The classes in the above three Java packages in J2SE 5.0 contain 85 direct references to classes, methods, or fields in 9 other packages. Those 9 packages are: java.lang.annotation, java.lang.reflect, java.math, java.net, java.nio, java.nio.channels, java.nio.charset, java.security and java.util.regex. If the search is limited to start with the 61 classes in these three Java packages that are actually mentioned in the JLS, including the 4 mentioned in java.lang.annotation, then there are just 30 direct references to elements in 5 other packages. Those packages are: java.lang.reflect, java.net, java.nio, java.nio.channels, and java.security.

The additional elements that are referenced are not "core" to the programming language. These references are in no way required by the JLS. They are, rather, due to the unconstrained choices made by the authors of the API specifications. As noted above the JLS mentions just 61 classes by name, but in most cases does not require them to have any particular methods or fields. The JLS does specify some methods, constructors, and fields for about 20 different classes, but

none of these specified elements references any element located outside of java.lang, java.io, or java.util. It would be wrong to say that a class or method is somehow "core" to the language simply because another class refers to it.

9. What cross-method, cross-class interdependencies exist at the implementation level in Java? Were any of these duplicated in the Android implementations? (The judge remembers no evidence on this point.)

All of the cross-method, cross-class interdependencies that are present in the APIs exist at the implementation level in Java and in Android for the 37 API packages that Google copied. Both parties agree the documentation and the implementation are drawn from the source code. The experts agree the SSO is the same. (*See* RT 1248:11-1249:25 (Mitchell) RT 2214:6-9 (Astrachan).)

In addition, many of the same cross-class relationships that exist at the declaration or method header level also exist at the implementation level. For example, a method might have a parameter list that includes a type defined in a different class. (*See* RT 602:18-603:4 (Reinhold) (discussing demonstrative method with class Color as a parameter).) It would make little sense for a method to take parameters as input—whether defined in the same class or another class—and then not use the parameters in the implementation.

With respect to implementation details that are unrelated to declarations and method headers, there are many cross-method, cross-class interdependencies between classes and packages in the Java implementation. Oracle did not submit any evidence at trial about the extent to which any of those additional interdepencies were copied by Android.

10. In Java, what interdependencies exist at the name/declaration level other than the inherited characteristics from the super class, interfaces, same class, etc.? Please explain.

At the level of method headers, dependencies upon other classes can occur in return and parameter types and in exception lists ("throws" clauses) (JLS 8.4, p. 210). A field declaration can refer to a class as the type of the field's values (JLS 8.3, p. 196). A class declaration places a class within a specific package and it can refer to a superclass and also to some implemented interfaces (JLS 8.1, p. 175), and similarly for interface declarations (JLS 9.1, p. 260). One can find a number of interdependencies reflecting original design choices in the declarations within

the Java platform. The choices of which parameters to include and the order in which to include them illustrate such interdependencies, because parameter types can come from other classes and packages. (*See* RT 602:18-603:4 (Reinhold).) The class java.net.Authenticator, for instance, has three different versions of a method called "requestPasswordAuthentication." (TX 610.2 at jdk-1_5_0-doc/docs/api/java/net/Authenticator.html.) The parameter list for the most detailed of these contains eight different parameters:

> *public static PasswordAuthentication requestPasswordAuthentication(String host,*
> *InetAddress addr,*
> *int port,*
> *String protocol,*
> *String prompt,*
> *String scheme,*
> *URL url,*
> *Authenticator.RequestorType reqType) {*

Id. With the exception of "int," which is a primitive type, each of the other parameters in this list is a type of class. InetAddress, URL, and Authentictor.RequestorType are classes within java.net, and String is specified in java.lang. Regardless of where in the specifications each parameter type is defined, however, requiring eight different input values in the order the developers chose to generate "[t]he username/password, or null if one can't be gotten," id., was a creative, nontrivial choice that could have been structured differently, for example by putting these parameters in a different method or class.

The list of exceptions a method throws also represents the expression of a structural choice. The API designer must choose which exceptions to handle within a given method and which code that calls the method will have to handle, or "catch" See TX 984 at 325-26. For example, the class java.security.cert.Certificate contains a method identified with the header:

> *public abstract void verify(PublicKey key, String sigProvider) throws CertificateException, NoSuchAlgorithmException, InvalidKeyException, NoSuchProviderException, SignatureException*

TX 610.2 at /docs/api/java/security/cert/Certificate.html. This method throws five different types of exceptions. The designers could have chosen to have the method throw a single type of exception or none.

11. With respect to the Seventh Circuit decision in *American Dental Association*:

(A) To what extent has it been adopted in the Ninth Circuit?

American Dental Association v. Delta Dental Plans Association ("ADA"), 126 F.3d 977 (7th Cir. 1997) has not been cited in a published opinion within the Ninth Circuit. In reaching its conclusion that the dental procedure taxonomy was protectable, however, the *ADA* court expressly relied on two Ninth Circuit decisions: *Practice Management Information Corp. v. American Medical Association,* 121 F.3d 516 (9th Cir. 1997) (*see ADA* at 978), and *Edwin K. Williams & Co. v. Edwin K. Williams & Co. East*, 542 F.2d 1053 (9th Cir. 1976) (*see ADA* at 981). In *Management Info.*, the Ninth Circuit affirmed the validity of the AMA's copyright in its catalog of medical procedure codes and descriptions ("Current Procedural Terminology" or "CPT"). 121 F.3d at 520. The CPT "identifies more than six thousand medical procedures and provides a five-digit code and brief description for each." *Id*. at 517. The court rejected plaintiff's argument that the CPT had entered the public domain when a federal health care agency adopted the CPT codes for administering its programs. *Id*. at 517-518. The court observed that "copyrightability of the CPT provides the economic incentive for the AMA to produce and maintain the CPT" and "invalidating its copyright on the ground that the CPT entered the public domain when [agency] required its use would expose copyrights on a wide range of privately authored model codes, standards, and reference works to invalidation." *Id*. at 518-19.

In the second case, *Edwin K. Williams*, the Ninth Circuit upheld the validity of copyrights on plaintiff's account books. 542 F.2d at 1060-61. The account books contained blank forms and instructions for the forms. *Id*. at 1060. One form, for example, was entitled "Stamp Control Promotion" and had a description that began "This section offers an easy method of accounting for Stamps or Promotion Items." *Id*. The court held that "the instructions and the blank forms constituted an integrated work entitled to copyright protection." *Id*. at 1061.

Similarly, in *Educational Testing Serv. v. Simon*, the court found that plaintiff's copyrights in standardized test questions and forms were infringed, noting that the "headings and layout of defendants' materials further make clear that their 'sample' questions were created through access to ETS's copyrighted questions provided orally through students who previously

took the test." 95 F. Supp. 2d 1081, 1088 (C.D. Cal. 1999). The court held that "ETS' selection of topics for a particular test form is an exercise requiring judgment and creativity, and is therefore copyrightable." *Id*. at 1089.

(B) If the taxonomy in that decision was protectable, why shouldn't Sun's hierarchical outline of packages, classes, methods for the 37 API packages be protectable (other than perhaps the three core packages?

The SSO for the 37 Java API packages is protectable. It is far more intricate and creative than the dental code taxonomy at issue in the *ADA* case. The fact and expert witnesses from both sides testifed that API design is highly creative and challenging. (RT 513:14-18 (Screven); 627:21-628:1 (Reinhold); 751:5-752:14 (Bloch); 2209:7-8 (Astrachan); TX 624 at 47 (Bloch presentation).) It took Dr. Reinhold and a team of engineers almost two years to develop java.nio and its related sub-packages. (RT 623:17-624:1.)

ADA recognized that taxonomies are creative works entitled to copyright protection. *ADA*, 126 F.3d at 979 (Facts do not supply their own principles of organization. Classification is a creative endeavor.) But unlike a taxonomy, the 37 API packages do not simply classify existing data. They result from the many creative design choices made by Sun's engineers from among numerous possibilities for addressing technical challenges. (RT 627:21-628:23 (Reinhold).) The SSO warrants protection under the principles expressed in the *ADA* decision. The java.lang, java.io, java.util packages packages are protectable as well, for the reasons discussed in section 6.

(C) Did *ADA* hold that the numbering system alone (apart from the description) was copyrightable?

Yes. *See ADA*, 126 F.3d at 979 ("The number assigned to any one of the three descriptions could have had four or six digits rather than five; guided tissue regeneration could have been placed in the 2500 series rather than the 4200 series; again any of these choices is original to the author of a taxonomy, and another author could do things differently."). The court recognized that "all three elements of the Code—numbers, short descriptions, and long descriptions, are copyrightable subject matter." *Id*.

12. With respect to the Ninth Circuit's decision in *Kapes*:

(A) *Kapes* stated "Whether CDN's selection and arrangement of the price lists is sufficiently original to merit protection is not at issue." 197 F.3d at 1256. If that was not issue, what, if anything, did *Kapes* expressly say about SSO?

The Ninth Circuit did not rule on the selection and arrangement of the price lists in *CDN v. Kapes*, 197 F.3d 1256, 1259 (9th Cir. 1999). But it did rely upon *Urantia Foundation v. Maaherra*, 114 F.3d 955, 959 (9th Cir. 1997) summarizing its holding as "the arrangement of divine revelations met the level of creativity required for copyright." In that case, the Ninth Circuit addressed the selection, arrangement and structure of papers purportedly containing the answers of divine beings to questions, which it treated as a protectable compilation:

> In this case, the Contact Commission may have received some guidance from celestial beings when the Commission posed the questions, but the members of the Contact Commission chose and formulated the specific questions asked. These questions materially contributed to the structure of the Papers, to the arrangement of the revelations in each Paper, and to the organization and order in which the Papers followed one another. We hold that the human selection and arrangement of the revelations in this case could not have been so "mechanical or routine as to require no creativity whatsoever." *Feist*, 499 at 362.

The *Urantia* case confirms that the Ninth Circuit will uphold selection, arrangement and structure in a written document, and that the standard for creativity it applies is a low one.

(B) In what sense were the "prices CDN creates" in *Kapes* a "compilation" within the meaning of the Copyright Act (see 197 F.3d at 1260, second col.).

Under 17 U.S.C. § 101, "A 'compilation' is a work formed by the collection and assembling of preexisting materials or of data that are selected, coordinated, or arranged in such a way that the resulting work as a whole constitutes an original work of authorship." The court agreed with the district court's finding that "the prices in CDN's guides are not facts, they are 'wholly the product of [CDN's] creativity.'" *Id.* (substitution in original). But at the same time, CDN derived its price estimates by looking at wholesale prices, dealer online networks, public auctions, private sales and other things, and exercising its own judgment in determining whether, and to what extent, to rely on each. *Id.* It appears the court concluded that, because the prices were based on data gathered from various sources, guides constituted a compilation of that data, even if infused with creativity. It referred to them as "compilations of data chosen and weighed with creativity." *Id.* at 1261.

(C) Didn't *Kapes* treat the coin prices as"compilations"? Please explain how this was done. Has Oracle abandoned the compilation argument herein?

Yes. The case refers to the coin prices as "compilations" multiple times. It analyzed the copyrightability of the prices as though they were compilations, citing the *Urantia* case and concluding that the process for arriving at the prices was sufficiently original to be copyrightable, even if the underlying factual data was not. *Id.* at 1260.

The evidence here does not support that the API packages were "formed by the collection of preexisting materials or of data[.]" 17 U.S.C. § 101. Rather, the API packages are works of authorship written from scratch. (*See, e.g.*, RT at 621:7-623:16 (Reinhold).) Some decisions have treated as "compilations" works that represent collections and assemblies of uncopyrightable or copyrightable elements, regardless of how those elements were created. *See, e.g, Merchant Transactions*, 2009 U.S. Dist. LEXIS 25663 at *46 ("coordination, selection, and arrangement of these field names, as well as the fields and field types themselves, formatting, and commenting, may be entitled to protection as a compilation so long as they 'are numerous enough and their selection and arrangement original enough that their combination constitutes an original work of authorship.'" (quoting *Satava*, 323 F.3d at 811). *See also id.* at *58 (looking to selection, coordination and arrangement of both unprotected works and protected source code).

In that sense, the Java API packages are protectable as compilations, as well as works of authorship, particularly to the extent the Court concludes, for example, that method declarations are not copyrightable themselves. Notwithstanding such a determination, the selection and arrangement of so many declarations in such an elaborate and extensive structure represents copyrightable subject matter under this line of authority.

(D) Was originality the only issue decided in *Kapes*?

No. *Kapes* repeatedly emphasized the creativity involved in estimating the prices. *See, e.g., id.* at 1261. *Kapes* also discussed the idea/expression dichotomy and the merger doctrine. The court held that "'the guiding consideration in drawing the line is the preservation of the balance between competition and protection reflected in the patent and copyright laws.'" *Id.* at 1262 (quoting *Herbert Rosenthal Jewelry Corp. v. Kalpakian*, 446 F.2d 738, 742 (9th Cir. 1971).

It concluded CDN could not copyright the idea of creating a wholesale pricing guide, but its prices warranted protection. The Court found this allowed CDN's competitors to create their own guides and compete, but still protected CDN's creation. *Id.* at 1262.

13. When discussing use of the SSO in the 37 API packages in Android to achieve "compatibility," what exactly are the parties referring to? Is it "compatibility" with programmers who write in the Java programming language? Or compatibility with pre-existing programs? If so, approximately what percent of pre-existing programs written for the Java platform are compatible with Android? Is it compatibility with the TCK? Or Java virtual machine? Or java compiler?

The record reflects two definitions of compatibility. In the Java-specific sense, an implementation is only "compatible" if it passes the TCK. (*See, e.g.*, RT 977:22-978:1 (Google admission) TX 610.1 (Specification License); Gupta at RT 2306:6-2307:14 ("Q. A licensee was required to pass the TCK, even if they didn't want to use the Java brand; is that right? A. Yes."); RT 294:8-21 (Ellison); RT 366:21-25 (Kurian); RT 636:4-637:2 (Reinhold).) It is undisputed that Android has never passed an Oracle TCK. (RT 984:22-24 (Lee).)

The more general notion of compatibility is that programs built for one platform will run on the other. Professor Mitchell explained how Android is incompatible in this way as well. RT 1331:16-1332:2, 2287:23-2288:5 (Mitchell). *See also* RT 1007:6-11, 1010:4-7 (Morrill); TX 383 at 8 (Android FAQ 048. "Does Android support existing Java apps? / A. No. / 049. Is Android Java compatible? / A. No."). Dan Bornstein testified ensuring Android would be "compatible" with Java in this sense "wasn't a goal of the project." (RT at 1783:15-22 (Bornstein).)

Oracle is aware of no metrics regarding the percentage of pre-existing programs written for the Java platform that are compatible with Android. The record refers to the fact that the byte code formats of Java and Android are different, such that a compiled Java program will not run, without modification, on Android, and vice versa, even if it only calls on the 37 copied packages. (*See* RT 2211:3-2212:2 (Astrachan); RT 2287:13-2288:5 (Mitchell).) More significantly, there are entire categories of incompatible programs because of packages that Google chose not to include. For example, Android lacks Java's java.awt.* and javax.swing.* packages, which provide user interface capability. Compare TX 610.2 at /jdk-1_5_0-doc/docs/api/overview-summary.html with TX 767 at packages.html. Instead Android uses its own set of graphics APIs,

including android.app.* and android.graphics.*. (*See id.*) This means that any Java program requiring use of a graphical user interface will not run on Android. Similarly, Android uses different APIs for sound (javax.sound.* packages are not present in package list in TX 767) and for image input/output (javax.imageio.* packages not present in package list in TX 767). So Android is incompatible in this regard as well. Even printing to paper (javax.print.*) is different between the two platforms. (*See id.*)

The fact that Android wrote its own APIs to handle many of these capabilities further demonstrates that Google's copying was by choice, not to achieve compatibility.

14. Inheritance does not exist among packages, only within a class. True? If not, why not?

True. While a number of packages have sub-packages, RT 1219:18-1220:5 (Mitchell), characteristics are inherited based on class-subclass relationships and interface relationships.

15. Inheritance is a characteristic of a class that results from the superclass-subclass feature of the Java operating systems. True? If not, why not?

True, with the caveat that Java is more accurately characterized as an application development platform rather than an operating system. A class can inherit directly from its superclass, and indirectly from that class's superclass, and so on. (*See* RT 588:5-11 (Reinhold); RT 1218:15-16, 1225:10-16 (Mitchell).) Interfaces can be used to relate different classes that share common characteristics that are located in different packages. (*See* RT 589:13-18, 590:5-23, 601:22-25 (Reinhold).) An interface can inherit directly from one or more superinterfaces and indirectly from those interfaces' superinterfaces, and so on. *See* RT 1219:16 (Mitchell). A class can directly inherit from only a single class, but can implement multiple interfaces.

16. *Assuming that a copyright protection does not extend to names, including fully qualified names, and assuming that copyright protection does not bar others from using identical input-output (argument-return) designations, such that Google was free to use the identical names and identical input-output designations, what more did Google allegedly copy from the 37 packages that is allegedly covered by copyright? Put differently, assuming Google was free to do the foregoing, to what extent was Android's SSO dictated by the rules of the basic programming language?*

Answering the first question, even if copyright protection does not extend to fully qualified names and input-output designations, Google still copied the following: (a) Types (but not names) in field declarations (*see* TX 984 at 221 (noting that field declarations must include

types); RT 602:22-603:2(Reinhold)); (b) Subclass/superclass relationships ("X extends Y") (RT 1218:15-19, 1225:10-16 (Mitchell)); (c) Interface implementation relationships ("X implements Z") (*See* RT 589:13-18, 590:5-23, 601:22-25 (Reinhold)); (d) Package/sub-package relationships (RT 2285:12-15 (Mitchell)); (e) Specification text on which Google "based" its implementing code (RT 2219:7-18 (Astrachan)); and (f) the original selection, structure, sequence, and organization of the above elements, including the names and the argument-return designations. Google copied all of these elements into Android. (RT 1248: 25-1249:25 (Mitchell).)

Moreover, as discussed above, even if individual names or argument-return combinations were not protectable, and even if Google had copied no more, a combination of unprotectable elements is still eligible for copyright protection. *Satava*, 323 F.3d at 811.

Answering the Court's second question, regardless of whether Google was free to do what the question assumes, Android's SSO was not dictated by the Java programming language. The only constraints Google faced were that it was obligated to follow the grammar of the Java programming language (TX 984 chapter 18) as well as the non-grammatical constraints of the language, *e.g.*, a class or interface cannot extend itself or be declared more than once. (*See id.*) Google was also obligated to include, in some form, the 60 or so classes discussed in response to question 7 above.

None of this dictates how to design the API. If it did, every API would be the same. The trial evidence showed that APIs performing similar functions were design very differently. (*See* Response to Question 4 above.) Google's expert admitted that Google could have designed its own APIs, and the irrefutable evidence in the record is that Google *did* design its own APIs for other packages. (RT 2213:8-19 (Astrachan).) The constructs of the programming language provide an opportunity for designing a rich and complex structure but do not require it. RT at 619: 13-23 (Reinhold) ("very little organization or structure" required for virtual machine and computer"); *See* also: RT 634:10-25 (Reinhold); RT 565:3-16 (Screven) ("there is nothing in the Java language specification that requires that there be a class named tree, but there is a class named tree within—within the standard APIs."); RT 1238:11-1240:20 (Mitchell) (API designer starts with a "clean slate").)

Dated: May 10, 2012

MORRISON & FOERSTER LLP

By: /s/ *Michael A. Jacobs*
Michael A. Jacobs

Attorneys for Plaintiff
ORACLE AMERICA, INC.